Suite++®

THE PLUM HALL VALIDATION SUITE FOR THE STANDARD C++ LANGUAGE

VERSION 2025a August 2025

Your Feedback is Valued

Please feel free to contact me with any issues, errors, omissions, thoughts, … concerning the test cases and infrastructure in the Plum Hall test suites. The software is constantly updated with new test cases and infrastructure improvements. A new distribution is released in the month of August every year. Please contact me by email: dougteeple at plumhall2b.com.

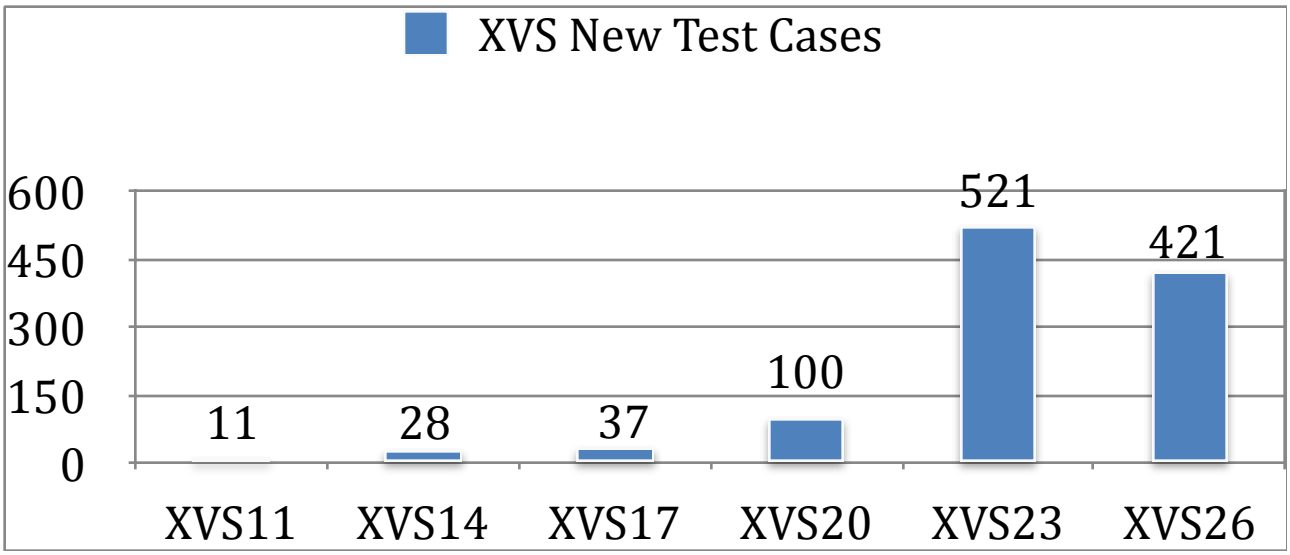
New in xvs25a:

This release has new test cases proposed for C++26 features.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23	ISO/IEC 14882:2023	202311L	C++2b
CXX26	ISO/IEC 9899:202y		Work In Progress

C++ Releases

There are 421 new test cases, documented in “coverage-c26.html”, in multiple directories predominantly in t26a.dir. The new test cases predominantly pertain to the proposed C++26 standards. The total number of test cases is now more than 7500, including positive, negative and undefined cases.



There are new test cases in a new category, which is by functionality:

c2y_cpp23.cpp
c2y_cpp26.cpp
c2y_MISRA23.cpp
c2y_ISO_26262.cpp
c2y_ub.cpp
c2y_threads.cpp
c2y_modules.cpp
c2y_lambda.cpp
c2y_coroutines.cpp
c2y_freestanding.cpp
c2y_embedded.cpp
c2y_annex_a.cpp
c2y_annex_b.cpp
c2y_annex_c.cpp
c2y_annex_d.cpp

Each test case tests for specific functionality, alluded to by the test name.

Running the Test Suite

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

envsuite(.bat) is a script which is basically a large case statement. The cases are settings for different compilers. Common compilers are available in the script. If your compiler is not represented, you can use the existing implementations as a guide. envsuite is called in such a way that it instantiates environment variables used by the build script to run the test cases.

flags.h is a header file included by each test case. It defines flags which determine the standards year to test against and features that should be tested for the corresponding standards year.

compiler-flags.h is another header included by each test case. It defines specific flags for each compiler. If your compiler is not represented, add it using existing cases as a guide. The flags are very restricting as shipped. The reason is that is the only way for the tests actually to return any results, particularly for newer standards years, for which few, if any of the features tested actually compile. So, after an initial run to get basic results, you may see that many test cases are not as skipped. Over time, remove the restraint flags to test newer standards features.

compiler-setup.bat is a Windows-only script that should be run after envsuite.bat to set up specific compiler environment variables. Modify as required (but don’t forget to run this script after envsuite.bat).

VERSION 2024a August 2024

Your Feedback is Valued

Please feel free to contact me with any issues, errors, omissions, thoughts, … concerning the test cases and infrastructure in the Plum Hall test suites. The software is constantly updated with new test cases and infrastructure improvements. A new distribution is released in the month of August every year. Please contact me by email: dougteeple at plumhall2b.com.

New in xvs24a:

This release has new test cases proposed for C++26 features.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23	ISO/IEC 14882:2023	202311L	C++2b
CXX26			Work In Progress

C++ Releases

C++ tests may now be checked as conforming to *16.4.2.5 Freestanding* as opposed to hosted implementations. as well as C++ freestanding tests as denoted in the document ISO/IEC DIS 14882:2023.

C++ also now adds tests for undefined behavior, which were previously only available for the C language. The tests are in conform/undeftests/u*.in. Software, especially FREESTANDING, should not contain any constructs which have undefined behavior as per the standards. ISO 26262 in particular has a focus on dealing with undefined/unspecified behavior of C/C++ and on preventing runtime errors. Obeying language standards is recommended by all current safety standards.

This release also adds new test cases for modules and coroutines in the directories t01a.dir and t01b.dir respectively. There are 37 new test cases, documented in “newcases-xvs23a-xvs24a.txt”, in multiple directories. These new test cases predominantly pertain to the C++23 and proposed C++26 standards. This version also adds support for CUDA files as compiled by the nvcc compiler. These cases must be compiled on CUDA-enabled hardware. See t00a.dir for examples. Initial support for modules tests can be found in t01a.dir. Directory t01b.dir contains new tests for coroutines.

New in xvs23a:

This release addresses many defect reports from customers in the 22a release and adds new tests for C++20 and C++23 features. As of this release support for older C++ versions prior to C++11 is dropped.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23			C++2b

C++ Releases

New Test Cases

This release also adds new test cases for modules and coroutines in the directories t01a.dir and t01b.dir respectively. There are 140 new test cases, documented in "newcases-xvs20b-xvs23a.txt", in multiple directories. These new test cases predominantly pertain to the C++20 and C++23 standards. This version also adds support for CUDA files as compiled by the nvcc compiler. These cases must be compiled on CUDA-enabled hardware. See t00a.dir for examples. Initial support for modules tests can be found in t01a.dir. Directory t01b.dir contains new tests for coroutines.

Bug Fixes

There were issues in unarchiving negtests, causing partial results and loss of sequencing. These issues have been fixed. A number of files had duplicate main's. A number of test cases were just skeletons, these have been filled in with the actual test case. Test cases with dynamic exceptions have been modified since ISO C++17 does not allow dynamic exception specifications. Test cases involving the *register* keyword ave been modified since *register* has been deprecated. Test cases involving the *volatile* keyword have been modified since *volatile* has been deprecated in many contexts.

The 23a update release represents 3+ years of test case bug fixing, infrastructure improvements, and new test cases for C17, C20, C23, C++20, and C++23, language and library enhancements. There are many improvements in enhancing the test cases themselves and also enhancing the reporting of the results, through the new html interfaces for reporting coverage, commentary on the intent of the test cases and improved standards conformance reporting.

New Test Cases

This release adds new test cases addressing:

- Char and string types,
- some support for modules,
- the “spaceship” operator <=> and
- char8_t, u8string and u8string_view
- concepts
- coroutines
- version header
- source_location
- format
- span
- ranges and range adapters
- syncstream
- init-statements and initializers in the range *for* statement
- new attributes: [[no_unique_address]], [[likely]], [[unlikely]]
- pack-expansions in lambda init-captures
- consteval
- constinit
- aggregate initialization using parentheses
- check compiler feature and attribute definitions.

Infrastructure

Installers are available on the PlumHall server for Linux (installPH.sh) and Windows (installPH.bat). These installers greatly simplify installing the PlumHall distributions in a standard layout as described below. Download the installers from the plumhall2b.com server and run the installers to create the default installations. The installers are customized for each customer:

```
ftp plumhall2b.com
Connected to plumhall2b.com.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
Name (plumhall2b.com:doug): OscarWilde1854
331 User OscarWilde1854 OK. Password required
Password: ************
passive
get installPH.sh
get installPH.bat
quit

~/installPH.sh --help
Download, check the MD5 hash and install the PlumHall test suites in $HOME/PlumHall/
Options:
  --cvs=<version>       : install CVS version e.g. --cvs=CVS002
  --xvs=<version>       : install XVS version e.g. --xvs=XVS002
  --lvs=<version>       : install CVS version e.g. --lvs=LVS002
  --compiler=<name>     : brief compiler name used to create directory
                          structure, e.g. gcc, edg, clang, etc
  --PW=<zip password>   : zip password for distribution
  --login=<login name>  : login name given in download instructions, e.g. techcontactname
  --username=<username> : suffix to user name given in download instructions,
                          e.g. techcontactname8345
  --keep                : do not delete existing directories before unpacking the distributions.
  --verbose             : chatty
  --help                : help me if you can…

Note: uses scp to securely copy the distributions from the plumhall2b.com server, zip to unpack the
distribution and md5sum to calculate the MD5 hash.
```

Executing the scripts will download your distributions and check the MD5 sums. If the sums do not match the scripts will exit with an error, please contact PlumHall should this occur. If the MD5 sums are correct then compressed files will be expanded and installed in the standard directory structure.

If you have trouble with the install scripts, you may enter the commands:

```
ftp plumhall2b.com
login: OscarWilde1854@plumhall2b.com
passwd: *********
passive
get installPH.sh or get installPH.bat
get xvs22a-XVS000.tar.gz
get xvs22a-XVS000.tar.gz.md5
get xvs22a-XVS000.zip
get xvs22a-XVS000.zip.md5
…
bye
```

If you did a manual download you may then run the installer script with the option *--nodownload* to unpack, check the MD5 signatures and create and populate the standard directory structures. The installer extracts into a directory named ~/PlumHall/ by default. Please ensure that the md5sum utility is available and verify that the MD5 sums compare.

The script may ask for your plumhall2b ftp password as part of the installation process.

The default folder naming convention is:

```
<Test Suite><PlumHall Release Year>-<Compiler Mnemonic>-c<Standards Year>/

e.g. xvs23a-gcc-c20/
```

For example, to create directories for each of the standards years C++17 and C++20, for compilers gcc and clang:

```
installPH.sh --stdyear=17 --stdyear=20 --compiler=gcc --compiler=clang
```

There are three main points of customization:

- flags.h for C/C++ version options,
- compiler-flags.h for compiler-specfic options and
- envsuite.sh (envsuite.bat) to customize the execution environment.

Some customization is possible by using envsuite command line options. For example: `envsuite.sh cc=g++-latest` sets the version of g++ to use. Type `envsuite.sh -h` for current arguments. Further customization requires editing envsuite.sh(.bat)

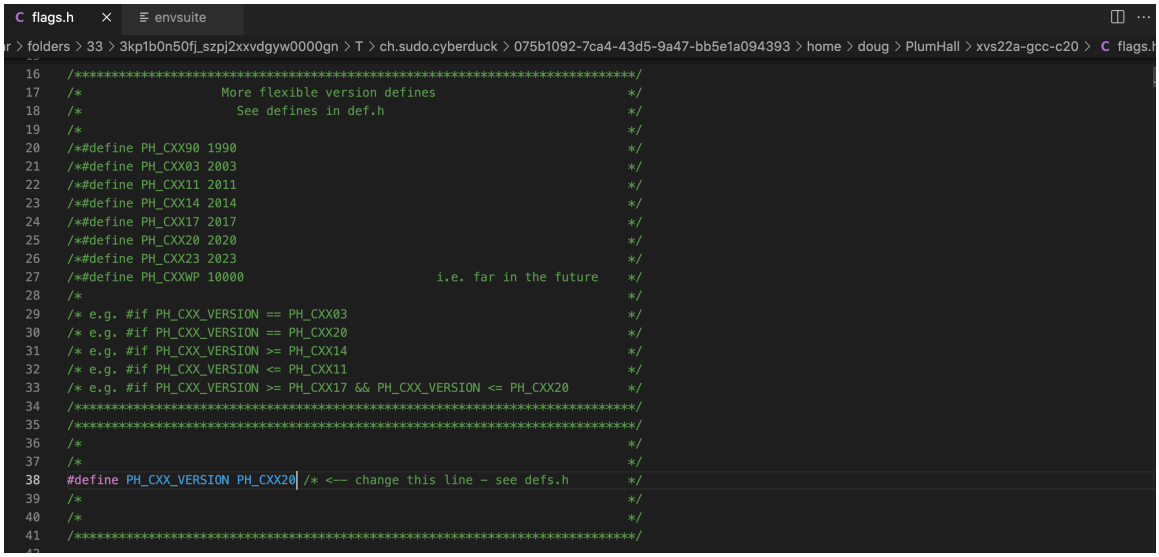
The `envsuite` script has been modified to more easily support a standard PlumHall directory structure and multiple compilers on the command line. The standard directory structure is:

```
~/PlumHall/xvs23a-<cc>-c20/          build directory
~/PlumHall/xvs23a                    source directory
~/PlumHall/xvs23a-<cc>-c20-setup/    setup directory updated by script save-setup

~/PlumHall/lvs23a-<cc>-c20/
~/PlumHall/lvs23a/
~/PlumHall/lvs23a-<cc>-c20-setup/

~/PlumHall/cvs23a-<cc>-c20/
~/PlumHall/cvs23a/
~/PlumHall/cvs23a-<cc>-c20-setup/
```

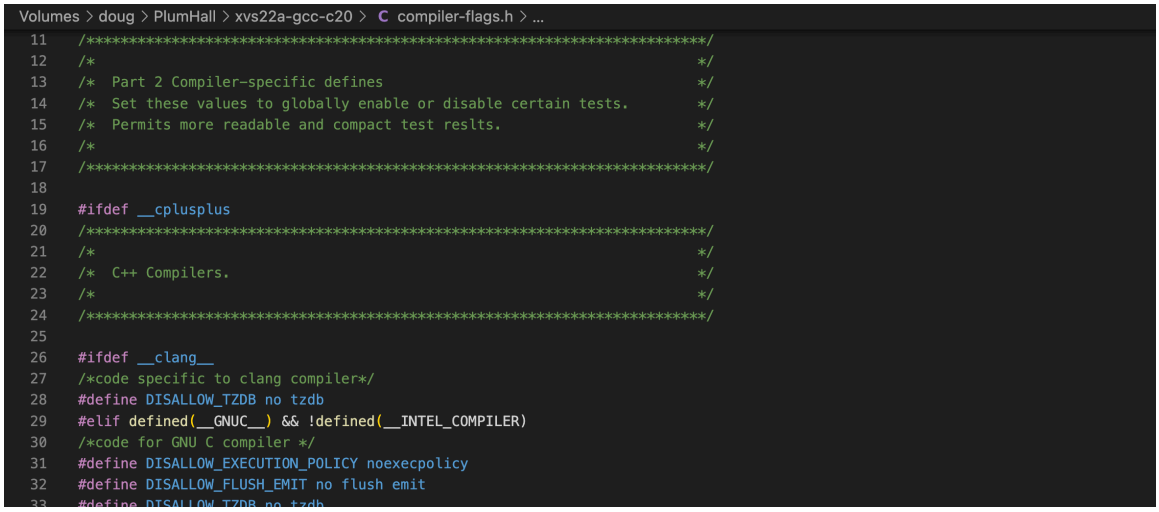
where <*cc*> is gcc or clang, or cl on Windows. The script `createDestination.sh` is available to create and populates these default directories, though the `installPH` scripts do this by default. It takes a command line argument cc=<gcc | clang | cl> to create different build directories for multiple compiler testing. The scripts take arguments cc=gcc or cc=gcc-latest or cc=clang-12 as examples. PH_CXX26 is set as the default release in flags.h and envsuite.

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

The file flags.h customizes for C++ and C standards releases version:



Customization of flags.h



Customization of compiler-flags.h

compiler-flags.h allows for setting flags specific to a particular compiler. These flags are often set to get around compile errors which prevent viewing overall results. For example lang.c and lib.c link in relevant test case object files. If a compile of a particular test fails, none of the results of the other tests can be seen.

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

```
#
# please edit your  save-setup  file to match your choice for PHDST
#
# envsuite --- environment for compiler ---
# $Revision: 26 2022-08-31 Copyright (c) 2086-2022, Plum Hall Inc. $

# defaults...
cc=gcc
compiler=${cc}
suite=xvs
relyear=22
suffix=a
release=${relyear}${suffix}
cvs=cvs
################################################################################
#                                                                              #
#                                                                              #
stdyear=20             # <-- change this line to build other releases         #
cstdyear=20            # <-- change this line to build other ctests           #
#                                                                              #
```

Customization of envsuite.sh

The release numbers in flags.h and envsuite MUST Be kept in sync. Customization of ensuite requires a detailed reading of the source of the script. Usually ensuite is used to set compiler and linker directories as required to build.

The build system itself has been enhanced. In prior releases adding a test case required hand editing multiple different makefiles and scripts. In this release this is no longer required, the makefiles and script automatically adjust to addition/deletion of test cases.

The build system did not adapt well to the new requirements imposed by C++ modules. The t01a.dir directory contains all the module test cases. The makefile and build script are customized to build modules in the style of gcc. At this time there is no support for building modules in the Microsoft cl.exe or clang styles.

In order to test modules and coroutines version 11 of gcc is required on Linux, and c++latest on Windows.

For example on Linux:

```
. ./envsuite cc=g++-latest
```

On Windows install the latest version of cl.exe and ensure that STD=c++latest is set in envsuite.bat.
A number of visualization tools have been added.

At the end of each buildmax build the following html files are created:

```
coverage-cxx20.html
commentary-cxx20.html
conform-ctests-cxx20.html
conform-cxx20.html
report-cxx20.html
```

The file conform-cxx.html and conform-ctests-cxx.html show a summary of successful tests and those with issues:

CSuite Conformance cxx20 xvs22a - gcc 11.0.1 20210 - Linux - Sat Jan 28 16:26:26 2023

ctests | C Conformance Tests.

negtests | Negative Tests - tests that should fail.

t01a.dir | Modules

t01b.dir | Coroutines

t02a.dir | raw strings, unicode, digrraphs

t03a.dir | ODR, declaratons

t03b.dir | friend declaratons, static, class template specification, name look up

t03c.dir | lookup, nested name spec, type-name, linkage

t03c.out	Compile error(s) 3432g12.cpp				Core 355. Global-scope :: in nested-name-specifier. if qualified-id starts with ::, name after global namespace - CXX11 - Implements N3259			
t03c.out	54	53	98%	0	1	0	0	See results in the output log.

t03d.dir | linkage, namespace, thread, static storage duration

t03e.dir | function alloc, dealloc, cv-qual, bool

t03e.out	54	54	100%	0	0	0	0	See results in the output log.

t04a.dir | rvalue, lvalue, cv-qualifiers

conform-cxx.html

The links to the source file, the output log, and the error log are all active and viewed as html.

The value in the **Expected** column is the number of test cases, where Expected = Actual + Errors + Faults + Aborts. The **Actual** column is the sum of the number of test results that matched expected values/behavior plus the number of skipped test cases. The value in the **Skipped** column is the number of skipped test cases. The value in the **Errors** column is the sum of the number of test cases that meet one of the following conditions:

- One or more unexpected values are returned in the test items.
- A compile error occurred, when the test file was compiled.
- An execution error occurred, when the test was executed.

The value in the **Abort** column is the number of test cases that and abort occurred. The value in the Faults column is the number of tests that meet one of the following conditions:

- An uncaught exception occurred when the test was executed.
- An internal error occurred when the test file was compiled.
- Unknown or unreported test results.

The links to the .out log file and .cpp source file help to quickly find what the issue is and where. The "t***.out" log filename in column 1 is a link to the actual output log of test result summaries for the entire test directory.

t03c.out

Output Log

```
st test *****
sed
c
l successful items in 3431e22c *****
l test case in 3431e22c *****
tected in 3431e22c *****
ections in 3431e22c *****


lum Hall Validation Suite for C++. License xvs22a-XVS002
or the use of employees of PlumHall.
two-mile radius of Plum Hall 67-1185 Mamalahoa Hwy Unit D104, PMB 372 Kamuela HI 96743 USA
re cloud site.
ct person at this site is Thomas Plum


 is provided by  Plum Hall Inc,
com. The software is copyrighted
o Plum Hall Inc, and by license is restricted to the
ite.  The reports it produces (such as this one) are
ion outside your organization.  Consult your license
rovisions.   We appreciate your ideas and questions.

f this software is executed after the date below, it
f-date. In that case, consult with your site Primary
```

t03.out.html

3432g12.cpp

Compile Error Report

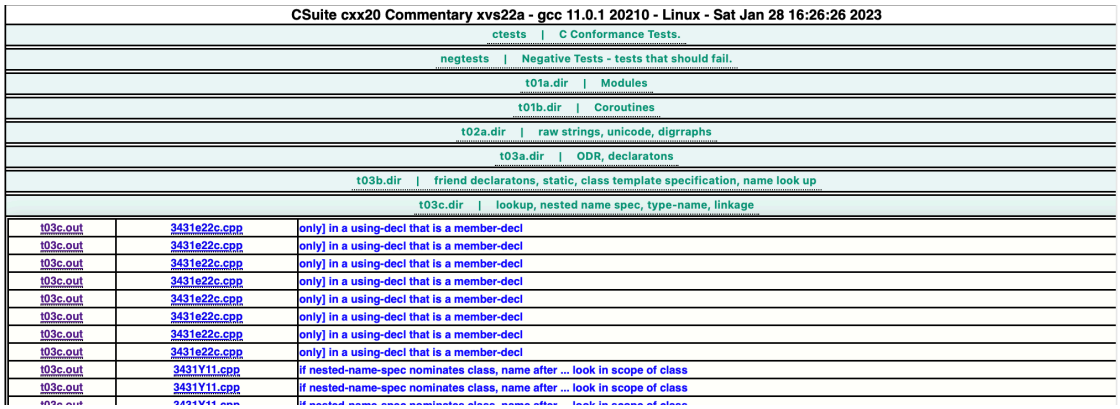
```
1.  3432g12.cpp:24:13  error: global qualification of class name is invalid before '{' token
2.  3432g12.cpp:46:12  error: variable 'A_ a' has initializer but incomplete type
3.
```

```
1.
2.
3.
4.  #include "defs.h"
5.
6.  /* 3432g12 Core 355. Global-scope :: in nested-name-specifier. if qualified-id starts with ::, name after :: in global nam
7.
8.  #if defined(SKIP3432g12)
```

t03c.cpp.html

The source file is linked to browse the test source file.

The `make-commentary` script creates an html file that shows a brief commentary of the purpose of each test case by folder name and test name:



CSuite cxx20 Commentary xvs22a - gcc 11.0.1 20210 - Linux - Sat Jan 28 16:26:26 2023

ctests | C Conformance Tests.

negtests | Negative Tests - tests that should fail.

t01a.dir | Modules

t01b.dir | Coroutines

t02a.dir | raw strings, unicode, digrraphs

t03a.dir | ODR, declaratons

t03b.dir | friend declaratons, static, class template specification, name look up

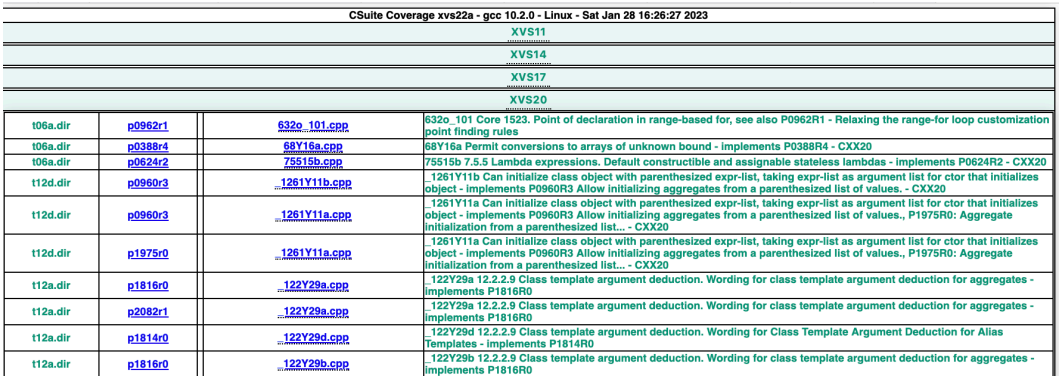
t03c.dir | lookup, nested name spec, type-name, linkage

t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431e22c.cpp	only] in a using-decl that is a member-decl
t03c.out	3431Y11.cpp	if nested-name-spec nominates class, name after ... look in scope of class
t03c.out	3431Y11.cpp	if nested-name-spec nominates class, name after ... look in scope of class

commentary-cxx.html

The filenames are links which will open the files for viewing in the html browser.

The `make-coverage` script generates the html file coverage-xvsxxa.html which shows for each C/C++ release, the Defect Report number, the directory test case file and a brief description of the Defect Report. This is useful to find which directories and test cases address a particular feature introduced by the Defect Report.



CSuite Coverage xvs22a - gcc 10.2.0 - Linux - Sat Jan 28 16:26:27 2023

XVS11

XVS14

XVS17

XVS20

t06a.dir	p0962r1	632o_101.cpp	632o_101 Core 1523. Point of declaration in range-based for, see also P0962R1 - Relaxing the range-for loop customization point finding rules
t06a.dir	p0388r4	68Y16a.cpp	68Y16a Permit conversions to arrays of unknown bound - implements P0388R4 - CXX20
t06a.dir	p0624r2	75515b.cpp	75515b 7.5.5 Lambda expressions. Default constructible and assignable stateless lambdas - implements P0624R2 - CXX20
t12d.dir	p0960r3	1261Y11b.cpp	_1261Y11b Can initialize class object with parenthesized expr-list, taking expr-list as argument list for ctor that initializes object - implements P0960R3 Allow initializing aggregates from a parenthesized list of values. - CXX20
t12d.dir	p0960r3	1261Y11a.cpp	_1261Y11a Can initialize class object with parenthesized expr-list, taking expr-list as argument list for ctor that initializes object - implements P0960R3 Allow initializing aggregates from a parenthesized list of values., P1975R0: Aggregate initialization from a parenthesized list... - CXX20
t12d.dir	p1975r0	1261Y11a.cpp	_1261Y11a Can initialize class object with parenthesized expr-list, taking expr-list as argument list for ctor that initializes object - implements P0960R3 Allow initializing aggregates from a parenthesized list of values., P1975R0: Aggregate initialization from a parenthesized list... - CXX20
t12a.dir	p1816r0	122Y29a.cpp	_122Y29a 12.2.2.9 Class template argument deduction. Wording for class template argument deduction for aggregates - implements P1816R0
t12a.dir	p2082r1	122Y29a.cpp	_122Y29a 12.2.2.9 Class template argument deduction. Wording for class template argument deduction for aggregates - implements P1816R0
t12a.dir	p1814r0	122Y29d.cpp	_122Y29d 12.2.2.9 Class template argument deduction. Wording for Class Template Argument Deduction for Alias Templates - implements P1814R0
t12a.dir	p1816r0	122Y29b.cpp	_122Y29b 12.2.2.9 Class template argument deduction. Wording for class template argument deduction for aggregates - implements P1816R0

coverage-cxx.html

Again the file names are links for convenient browsing of the test case suite. All of these html documents are produced dynamically from the source as the last steps in the `buildmax` script.

The make-report script generates a table showing all files with the associated commentary:

HTML Report of Commentary from All Sources - cxx20

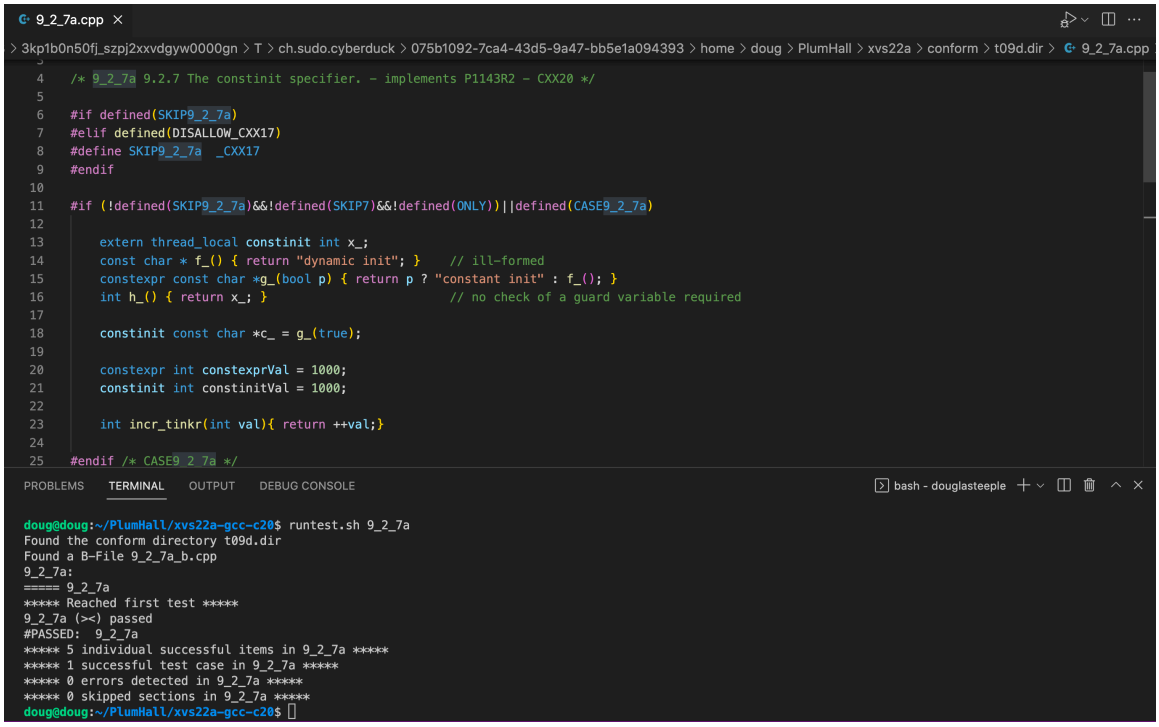
xvs-22a

conform

t12b.dir

_124Y63c.cpp	_124g83c bases and members destroyed in reverse order of ctors
_124Y71b.cpp	_124g91 can decl dtor as virtual or pure virtual; ...
_124Y63e.cpp	_124g83e bases and members destroyed in reverse order of ctors
_124Y33aa.cpp	_124g52 union-like class, non-trivial dtor
_124k61.cpp	a dtor that is defaulted and not defined as deleted
_124i13.cpp	id-expr is ~class-name ... names the current instantiation
_124Y_121.cpp	_124g_131 in explicit dtor call, dtor name appears as ...
_124Y71a.cpp	_124g91a can decl dtor as virtual or pure virtual; ...

There is a new script `runtest.sh(.bat)` which, given a test identifier, will find that file in the source directory and execute just that test. It is useful for debugging test cases. Here is an example of usage in Visual Studio:



C++23 Language Features

Feature	FileName	Addressed	Test Cases
Literal suffix for (signed) size_t	P0330R8		t05h.dir/5_1323a.cpp
Make () more optional for lambdas	P1102R2		t06a.dir/75515c.cpp
if consteval	P1938R3		t08a.dir/8_5_2c.cpp
Removing Garbage Collection Support	P2186R2		lvs: t203.dir/_207_137a1a.cpp
DR: C++ Identifier Syntax using Unicode Standard Annex 31	P1949R7		
DR: Allow Duplicate Attributes	P2156R1		t09.dir/9_121a.cpp
Narrowing contextual conversions in static_assert and constexpr if	P1401R5		t08a.dir/8_5_2d.cpp
Trimming whitespaces before line splicing	P2223R2		t05b.dir/5221a.cpp
Make declaration order layout mandated	P1847R4		t09a.dir/1121a.cpp
Removing mixed wide string literal concatenation	P2201R1		t05a.dir/5135a.cpp
Deducing this	P0847R7		t06a.dir/7552a.cpp
auto(x) and auto{x}	P0849R8		t07d.dir/76141a.cpp
Change scope of lambda trailing-return-type	P2036R3		t06a.dir/75515d.cpp
#elifdef and #elifndef	P2334R1		t16a.dir/_1527a.cpp
Non-literal variables (and labels and gotos) in constexpr functions	P2242R3		t05h.dir/57p1a.cpp
Consistent character literal encoding	P2316R2		t16a.dir/_1527b.cpp
Character sets and encodings	P2314R4		
Extend init-statement to allow alias-declaration	P2360R0		t07c.dir/_9_91a.cpp
Multidimensional subscript operator	P2128R6		t08b.dir/_2232a.cpp

C++20 Language Features

Feature	FileName	Addressed	Test Cases
Allow lambda-capture [=, this]	P0409R2		negtests/m05.in, t05a.dir/5452o22.cpp
__VA_OPT__	P0306R4 P1042R1		t16a.dir/_163o5a_s.cpp
Designated initializers	P0329R4		negtests/m08.in, t08b.dir/86o1a.cpp, t08b.dir/86o1a.cpp, t13e.dir/_133315o21.cpp
template-parameter-list for generic lambdas	P0428R2		t05a.dir/545o03.cpp, t05a.dir/545o03.cpp
Default member initializers for bit-fields	P0683R1		negtests/m09.in, t09a.dir/92o81.cpp, t09a.dir/92o81.cpp
Initializer list constructors in class template argument deduction	P0702R1		t13d.dir/_13318n1a.cpp
const&-qualified pointers to members	P0704R1		t05g.dir/554o62.cpp
Concepts	P0734R0		negtests/m05.in, negtests/m14.in, t05a.dir/544o31.cpp, t05a.dir/547o1a_s.cpp, t14a.dir/_141o1a.cpp, t14a.dir/_141o1b.cpp, t14a.dir/_141o1c.cpp, t14a.dir/_141o1d.cpp, t14b.dir/_143o1a.cpp, t14b.dir/_145o1a.cpp, t14b.dir/_145o1b.cpp, t14b.dir/_145o1c.cpp, t14b.dir/_145o1d.cpp, t14b.dir/_145o1e.cpp, t14b.dir/_145o1f.cpp, t14b.dir/_145o1g.cpp, t14b.dir/_147o1a.cpp, t14b.dir/_147o1b.cpp
Lambdas in unevaluated contexts	P0315R4		negtests/m05.in
Three-way comparison operator	P0515R3		t12e.dir/_1292o1a.cpp
DR: Simplifying implicit lambda capture	P0588R1		negtests/m09.in, negtests/m12.in, t05a.dir/5441n29.cpp
init-statements for range-based for	P0614R1		t06a.dir/654o11.cpp
Default constructible and assignable stateless lambdas	P0624R2		t05a.dir/75515b.cpp
const mismatch with defaulted copy constructor	P0641R2		t08b.dir/842o12.cpp
Access checking on specializations	P0692R1		t09a.dir/9Y25.cpp
ADL and function templates that are not visible	P0846R0		negtests/m14.in, t14b.dir/_142o22.cpp
DR: Specify when constexpr function definitions are needed for constant evaluation	P0859R0		negtests/m14.in, t14g.dir/_1481n7a.cpp
Attributes [[likely]] and [[unlikely]]	P0479R5		t07d.dir/7_117p1a.cpp
Make typename more optional	P0634R3		negtests/m14.in
Pack expansion in lambda init-capture	P0780R2		t05a.dir/5452p_172.cpp
Attribute [[no_unique_address]]	P0840R2		t03d.dir/362p1a.cpp
Conditionally Trivial Special Member Functions	P0848R3		
DR: Relaxing the structured bindings customization point finding rules	P0961R1		t08b.dir/85n3d.cpp
DR: Relaxing the range-for loop customization point finding rules	P0962R1		t06a.dir/632o_101.cpp, t06a.dir/654n1a.cpp
DR: Allow structured bindings to accessible members	P0969R0		t08b.dir/85n4c.cpp
Destroying operator delete	P0722R3		t09b.dir/936p1a.cpp
Class types in non-type template parameters	P0732R2		t14b.dir/_141p4a.cpp
Deprecate implicit capture of this via [=]	P0806R2		t05a.dir/5452p1a.cpp
explicit(bool)	P0892R2		t07a.dir/712p3a.cpp
Integrating feature-test macros	P0941R2		t16a.dir/_161p4a.cpp
Prohibit aggregates with user-declared constructors	P1008R1		t06a.dir/631p1a.cpp
constexpr virtual function	P1064R0		t07b.dir/715p4a.cpp
Consistency improvements for comparisons	P1120R0		t05h.dir/568p1a.cpp
char8_t	P0482R6		t02a.dir/2_134Y12e.cpp, t03d.dir/371p2a.cpp
std::is_constant_evaluated()	P0595R2		t05h.dir/57p1a.cpp
constexpr try-catch blocks	P1002R1		t16a.dir/_16_4_6_13a.cpp

Immediate functions (consteval)	P1073R3		t05h.dir/57p6a.cpp
Nested inline namespaces	P1094R2		t07d.dir/771p1a.cpp
Yet another approach for constrained declarations	P1141R2		t16a.dir/_16_31p2a.cpp
Signed integers are two's complement	P1236R1		t03d.dir/371p1a.cpp
dynamic_cast and polymorphic typeid in constant expressions	P1327R1		t05h.dir/5_19p2_15.cpp
Changing the active member of a union inside constexpr	P1330R0		t05h.dir/57p2_10.cpp
Coroutines	P0912R5		t01b.dir/*
Parenthesized initialization of aggregates	P0960R3		negtests/m05.in//, negtests/m08.in//, negtests/m08.in, negtests/m12.in//, t05a.dir/51Y51.cpp, t05c.dir/11_10_2a.cpp, t05c.dir/523m1a.cpp, t07b.dir/7162c41.cpp, t12d.dir/_1261Y11a.cpp, t12d.dir/_1261Y11b.cpp, t14c.dir/_1432g13e.cpp
DR: Array size deduction in new-expressions	P1009R2		t09a.dir/9345a.cpp
Modules	P1103R3		t01a.dir/*
Stronger Unicode requirements	P1041R4 P1139R2		t02a.dir/2_134Y12b.cpp, t02a.dir/2_134Y12c.cpp, t02a.dir/2_134Y12.cpp
<=> != ==	P1185R2		negtests/m05.in, negtests/m07.in, t05h.dir/568p1a.cpp, t09d.dir/9_113r1a.cpp, t11d.dir/_11_113r1a.cpp, t12e.dir/_1292o1a.cpp, t13b.dir/_13312p8a.cpp, t13b.dir/_13312p8a.cppauto, t14b.dir/_141p4a.cpp
DR: Explicitly defaulted functions with different exception specifications	P1286R2		negtests/m08.in, t09a.dir/9236r1a.cpp
Lambda capture and storage class specifiers of structured bindings	P1091R3 P1381R1		t09d.dir/9_6_1a.cpp,t09d.dir/9_6_2a.cpp,t09d.dir/9_6_3a.cpp, t09d.dir/9_6_4a.cpp
Permit conversions to arrays of unknown bound	P0388R4		to6a.dir/68Y16a.cpp
constexpr container operations	P0784R7		t09d.dir/9_2_6b.cpp
Deprecating some uses of volatile	P1152R4		multiple
constinit	P1143R2		t05h.dir/57p6a.cpp, t16a.dir/_161p4a.cpp
Deprecate comma operator in subscripts	P1161R3		negtests/m07.in #389
[[nodiscard]] with message	P1301R4		negtests/m07.in, negtests/m07.in, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp//, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp, t16a.dir/_161p4a.cpp
Trivial default initialization in constexpr functions	P1331R2		t09d.dir/9_2_6a.cpp, t09d.dir/9_2_6a.cpp
Unevaluated asm-declaration in constexpr functions	P1668R1		t07d.dir/77r1d.cpp
using enum	P1099R5		t09c.dir/972Y3.cpp
Synthesizing three-way comparison for specified comparison category	P1186R3		negtests/m05.in, negtests/m07.in, t05h.dir/568p1a.cpp, t09d.dir/9_113r1a.cpp, t11d.dir/_11_113r1a.cpp, t12e.dir/_1292o1a.cpp, t13b.dir/_13312p8a.cpp, t13b.dir/_13312p8a.cppauto, t14b.dir/_141p4a.cpp
DR: [[nodiscard]] for constructors	P1771R1		negtests/m07.in, negtests/m07.in, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp//, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp, t07d.dir/767k11.cpp, t16a.dir/_161p4a.cpp
Class template argument deduction for alias templates	P1814R0		negtests/m07.in//, negtests/m07.instruct, negtests/m14.in//, t14b.dir/_141g_111.cpp, t14b.dir/_143c61d.cpp, t14b.dir/_143o1a.cpp, t14c.dir/_1433g11c.cpp, t14c.dir/_1433g11.cpp
Class template argument deduction for aggregates	P1816R0 P2082R1		t12a.dir/_122Y29a.cpp
DR: Implicit move for more local objects and rvalue references	P1825R0		t11d.dir/_1195r1a.cpp
Allow defaulting comparisons by value	P1946R0		t05c.dir/11_10_2a.cpp
Remove std::weak_equality and std::strong_equality	P1959R0		t12e.dir/_1292o1a.cpp, t05h.dir/568p1a.cpp

Inconsistencies with non-type template parameters	P1907R1		negtests/m14.in//, t14b.dir/_141p4a.cpp, t14c.dir/_1442c11b.cpp, t14c.dir/_1442c11cb.cpp, t14d.dir/_14651c31.cpp, t14e.dir/_14623i21bd.cppint, t14e.dir/_14723c21.cppint, t14g.dir/_14821k12.cpp, t14g.dir/_14825g_161d.cpp, t14g.dir/_14825m_131a.cpp
DR: Pseudo-destructors end object lifetimes	P0593R6		negtests/m05.in, negtests/m05.in//, negtests/m05.inchar, t03c.dir/343Y51.cpp, t05b.dir/52c1_12.cpp, t05b.dir/52c1_13.cpp, t05b.dir/52c1_23.cpp, t05b.dir/52c1_25.cpp, t05b.dir/52f1_26.cpp, t05c.dir/523Y22.cpp, t05c.dir/524Y25.cpp, t05c.dir/524Y26.cpp, t13h.dir/_1356Y13d.cpp
DR: Converting from T* to bool should be considered narrowing	P1957R2		t07c.dir/7315Y.cpp

C++17 Language Features

Feature	Paper	Addressed	Test Cases
New auto rules for direct-list-initialization	N3922		t09d.dir/9_2_9_6_2a.cpp
static_assert with no message	N3928		t07a.dir/7j1_15da.cpp
typename in a template template parameter	N4051		t14a.dir/_141j13.cpp
Removing trigraphs	N4086		t02a.dir/22_31a.cpp, negtests/m02.in
Nested namespace definition	N4230		t07c.dir/731k_101.cpp
Attributes for namespaces and enumerators	N4266		t07c.dir/72j2_11a.cpp, t07c.dir/72j2_11b.cpp, t07c.dir/731j19b.cpp, t07c.dir/731j19c.cpp
u8 character literals	N4267		t02a.dir/2_133k31.cpp
Allow constant evaluation for all non-type template arguments	N4268		t05h.dir/5_20j41.cpp, t14c.dir/_1432j11.cpp
Fold Expressions	N4295		t05b.dir/513k11.cpp
Unary fold expressions and empty parameter packs	P0036R0		negtests/m14.in
Remove Deprecated Use of the register Keyword	P0001R1		negtests/m02.in
Remove Deprecated operator++(bool)	P0002R1		t13h.dir/_136Y41.cpp
Make exception specifications part of the type system	P0012R1		t05a.dir/512k72.cpp, t05a.dir/5k_132.cpp, t08b.dir/853k42fb.cpp
Aggregate initialization of classes with base classes	P0017R1		t09d.dir/9_4_2a.cpp
__has_include in preprocessor conditionals	P0061R1		t16a.dir/_161k0a.cpp, t16a.dir/_161k0a.cpp
DR: New specification for inheriting constructors (DR1941 et al)	P0136R1		t07c.dir/733k_151.cpp, t12d.dir/_1263k11a.cpp, t12d.dir/_1263k11a.cpp, t12d.dir/_1263k11b.cpp
Lambda capture of *this	P0018R3		negtests/m05.in, t05a.dir/512k92.cpp
Direct-list-initialization of enumerations	P0138R2		negtests/m08.in, t08b.dir/854k339.cpp
constexpr lambda expressions	P0170R1		t03e.dir/39k_1052.cpp, t05a.dir/512k21b2.cpp, t05a.dir/512k21b.cpp, t05a.dir/512k68c.cpp, t05a.dir/512k74.cpp, t05a.dir/512k74d.cpp
Differing begin and end types in range-based for	P0184R0		t16a.dir/_2431a
[[fallthrough]] attribute	P0188R1		negtests/m07.in, t07d.dir/765k11.cpp
[[nodiscard]] attribute	P0189R1		negtests/m07.in, t07d.dir/767k11.cpp
[[maybe_unused]] attribute	P0212R1		negtests/m07.in, t07d.dir/766k11.cpp, t07d.dir/766k21.cpp
Hexadecimal floating-point literals	P0245R1		t02a.dir/2_138k07_s.cpp
Using attribute namespaces without repetition	P0028R4		t07d.dir/761g12.cpp
Dynamic memory allocation for over-aligned data	P0035R4		t03e.dir/374m24b_s.cpp, t03e.dir/374m24_s.cpp, t03e.dir/374m24_s.cpp, t05g.dir/535m_11a.cpp
Class template argument deduction	P0091R3		negtests/m13.in, t07d.dir/7175m1a.cpp, t13d.dir/_13318m1a.cpp, t13d.dir/_13318m1b.cpp, t14g.dir/_149m1a.cpp
Non-type template parameters with auto type	P0127R2		negtests/m14.in, t14c.dir/_1432m21.cpp, t14g.dir/_14825m_131a.cpp
Guaranteed copy elision	P0135R1		negtests/m05.in, negtests/m07.in, negtests/m08.in, negtests/m12.in, negtests/m13.in, t04a.dir/44m1a.cpp, t06a.dir/663g21a.cpp, t07d.dir/7172m51.cpp, t08b.dir/86m_1761.cpp, t12a.dir/_122m1a.cpp, t12a.dir/_122m1b.cpp, t12a.dir/_122m1c.cpp
Replacement of class objects containing reference members	P0137R1		t05h.dir/57j71.cpp, t05h.dir/59m31.cpp
Stricter expression evaluation order	P0145R3		
Structured Bindings	P0217R3		t06a.dir/631r1a.cpp, t08b.dir/85m1a.cpp, t08b.dir/85m1b.cpp, t08b.dir/85n1a.cpp, t08b.dir/85n1e.cpp, t08b.dir/85n3d.cpp, t08b.dir/85n4c.cpp, t09d.dir/9_118r1a.cpp
Ignore unknown attributes	P0283R2		t07d.dir/7_117p1b
constexpr if statements	P0292R2		t08a.dir/8_5_2a.cpp
init-statements for if and switch	P0305R1		t08a.dir/8_5_2b.cpp
Inline variables	P0386R2		t09d.dir/9_2_8a.cpp
Removing dynamic exception specifications	P0003R5		negtests/m08.in, negtests/m15.in, t03a.dir/337g11.cpp, t03b.dir/341Y71.cpp, t05g.dir/537i32_s.cpp//, t08a.dir/835g11.cpp, t08a.dir/8Y44b.cpp, t12a.dir/_121Y12.cpp, t12b.dir/_124Y12.cpp, t13c.dir/_133112i21b.cpp, t15b.dir/_154g91_s.cpp, t15b.dir/_154j51.cpp, t15b.dir/_154Y_112b_s.cpp, t15b.dir/_154Y_121.cpp, t15b.dir/_154Y_121.cpp, t15b.dir/_154Y_132.cpp, t15b.dir/_154Y_132.cpp, t15b.dir/_154Y15e.cpp, t15b.dir/_154Y15f.cpp, t15b.dir/_154Y23a.cpp, t15b.dir/_154Y81a_s.cpp, t15b.dir/_154Y81a_s.cpp, t15c.dir/_1552Y11_s.cpp, t15c.dir/_1552Y11_s.cpp, t15c.dir/_1552Y21_s.cpp, t15c.dir/_1552Y21_s.cpp, t15c.dir/_1552Y23b_s.cpp, t15c.dir/_1552Y23c_s.cpp, t15c.dir/_1552Y_99a_s.cpp, t15c.dir/_1552Y_99a_s.cpp
Pack expansions in using-declarations	P0195R2		negtests/m07.in, t07c.dir/733m1a.cpp
DR: Matching of template template-arguments excludes compatible templates	P0522R0		negtests/m14.in, t14c.dir/_1433m31a.cpp, t14c.dir/_1433m31b.cpp

C++14 Language Features

Feature	Paper	Addressed	Test Cases
Tweaked wording for contextual conversions	N3323		t05e.dir/534i62e.cpp, t05g.dir/535Yi17b.cpp, t05g.dir/535Yi17.cpp, t05h.dir/5_19i61a.cpp, t05h.dir/5_19i61c.cpp, t05h.dir/5_19i61d.cpp, t06a.dir/642i22.cpp
Binary literals	N3472		t02a.dir/2_142i11b.cpp
decltype(auto), Return type deduction for normal functions	N3638		t05a.dir/5452p_172.cpp: t07b.dir/7162i21.cpp t07b.dir/7162i21.cpp:decltype(auto) t07b.dir/7164i_124b.cpp:template<class t07b.dir/7164i44.cpp:decltype(auto) t07b.dir/7164j7_11.cpp: t07b.dir/7164k771.cpp t07b.dir/7164k771.cpp
Initialized/Generalized lambda captures (init-capture)	N3648		t05a.dir/512i1_11b.cpp t05a.dir/512i_115b.cpp t05a.dir/512i_115c.cpp t05a.dir/512i_115.cpp t05a.dir/5452p_172.cpp t14b.dir/_141g_15a.cpp
Generic lambda expressions	N3649		t05a.dir/545o03.cpp
Variable templates	N3651		t04a.dir/431r1a.cpp t12e.dir/_128r1a.cpp t14a.dir/_14i19a.cpp
Extended constexpr	N3652		t05h.dir/5_19i2_26b.cpp: t07b.dir/715c61.cpp: t07b.dir/
Aggregates with default member initializers	N3653		t03a.dir/337c11a.cpp t03a.dir/337c11b.cpp t05a.dir/511e21.cpp t08b.dir/85c11.cpp t08b.dir/85c14b.cpp t09a.dir/92c0_11c.cpp t09a.dir/92c0_11.cpp t09a.dir/92c0_11e.cpp t09a.dir/92c22c.cpp t09a.dir/92c22e.cpp t09a.dir/92c22f.cpp t09a.dir/92c51.cpp t12d.dir/_1262c81c.cpp t12e.dir/_127c42d.cpp t12e.dir/_127c52a.cpp t12e.dir/_127c52b.cpp
Omitting/extending memory allocations	N3664		t05f.dir/534Y81.cpp, negtests/m07.in, t12e.dir/_129j12aa
[[deprecated]] attribute	N3760		t07c.dir/731j19c.cpp, t07c.dir/72j2_11b.cpp, t07c.dir/731j
Sized deallocation	N3778		t09b.dir/936p1a.cpp
Single quote as digit separator	N3781		t02a.dir/2_148c42b_s.cpp

1. Historical Overview

NOTE: SOME OF THE INFORMATION BELOW IS RETAINED AS A HISTORICAL REFERENCE.

For example, while CXX03 may be referenced, it is no longer supported because Microsoft's cl compiler only supports versions as far back as CXX14, thus the suites do not support any version prior to CXX11.

***Suite*++®**, the Plum Hall Validation Suite for C++, is a set of C++ programs for testing and evaluating a C++ language implementation.

This manual will explain how each section of the suite works, how to configure the tests for your system, and what assumptions are made about previous sections. The examples will illustrate the use of ***Suite*++**, and also demonstrate how some of the sections work.

If you have never used ***Suite*++** before, you need to read all of this manual. This is a large, extremely configurable suite of test programs. It can provide you with a very powerful testing environment, but it usually takes several hours to set up the first time. If you get stuck, or have problems, don't hesitate to call Plum Hall for technical support. We want you to succeed with this project.

Otherwise, the process should be familiar from your previous work with Plum Hall suites.

New in xvs20a and in xvs19a:

Each subdirectory, such as t02a.dir, provided one large file, such as t02a.cpp, which collected together all or most of the tests in this section. We have dropped support for the Suite++ feature of providing files such as xvs02a. We have reluctantly concluded that we have no way of implementing this feature.

New in xvs18a:

Contrary to our expectations a few years ago, we have had to accommodate CXX17 and CXXWP. The code for CXX17 is 'n' and the code for CXXWP is 'o'. There are some decisions of the Core Group of SC22/WG21 that are Defect Reports for C++17, and others that only affect C++20.

New in xvs17a:

The requirements of ISO/IEC editors have caused the chapters ("clauses") in the C++ Standard to be re-numbered. What was originally clause 17 is now clause 20, etc. Fortunately, the offset is a constant (three).

For now, only the members of the C++ standards committee are affected by this change, but eventually everyone will see this offset.

Plum Hall has not changed the testcase numbering system; those of you consulting the most recent drafts will need to subtract three from the clause number. NOTE: since cxx17 the numbering scheme has gotten much cloudier, please use the coverage report as discussed above to correlate test cases to defect reports.

1.1 License

Please take the time to read the license that your organization has signed. It is a legal document, and the restrictions apply to any persons using the product.

Here is a brief summary:

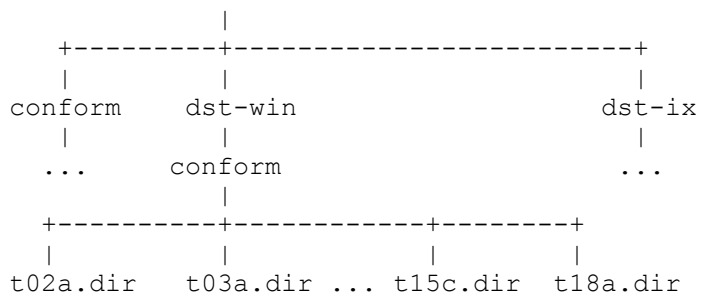
- You may use ***Suite*++** on any machine within a 2-mile radius of your Designated Site.

- Your Management Contact person, or anyone designated by the Management Contact, may call Plum Hall for consultation and advice.
- You need to notify us if you designate a new Management Contact, or plan to change your Designated Site, or plan to change your company's name.
- ***Suite***++ is proprietary, confidential, copyrighted software. You must protect its confidentiality with the same procedures you use to protect your own company's confidential information.
- You may not disclose the detailed results of running ***Suite***++, except as permitted in the License.
- You may not take any form of copies of ***Suite***++ away from the Designated Site.

1.2 Technical Overview

The normative tests of ***Suite***++ are found underneath one directory named **`CONFORM`**; these are positive tests for basic conformance with the Standard. (This section provides coverage for C++ analogous to the **`LANG`** tests of the Plum Hall C Validation Suite.)

Tools for use in different "destination" directories are provided in the directory trees named **`dst-win`** and **`dst-ix`**. Each of these contains a subtree that matches the structure of the source directories in **`CONFORM`**. Each subtree contains subdirectories for the various specialized tests of ***Suite***++, named **`t02a.dir`**, **`t03a.dir`**, etc. Thus, the components of ***Suite***++ are arranged in a directory tree something like this:



All the configurable files are now found in the "destination root" directory. Your compile scripts need to use **`$PHDST`** (or **`%PHDST%`**) in their search-path for header files in order for the compiler to find the configurable headers.

Also, we define the compiler's name as **`$PHCC`** (or **`%PHCC%`**) in the **`envsuite`** scripts. Configure this to the name of your compiler's executable file (e.g. **`mycc`**).

A useful feature of ***Suite***++ allows you to record the reasons for each compile-time skipped case or run-time failure. In your **`flags.h`** file, you can add a definition to some compile-time flags, such as

```
#define SKIP525Y1_11 our parser error
#define FAIL_261Y11  Plum Hall bug?
```

Once you've categorized your skips in this way, the strings you defined will show up in the execution output, something like this:

```
#SKIPPED 525Y1_11 (>our parser error<)
#FAILED _261Y11  (>Plum Hall bug?<)
```

And the "unexpected" skips and fails will show up with the distinctive string **`"(><)"`** attached to each "unexpected" skip or fail. This makes it much easier to re-run the test suite after you've made compiler changes, because you can quickly search for the **`"><"`** string in the output to see if any new failures have appeared.

You probably will need to put some **SKIP** flags into your **flags.h** file to skip test cases that prevent you from building and executing the **CONFORM** programs.

In ***Suite***++, we have also provided a simpler way of determining the **SKIP** and **FAIL** flags. Each subdirectory, such as **t02a.dir,** the subdirectory also provides individual files, **2_10Y04.cpp**, **2_10Y11.cpp**, etc., each of which contains only one specific test case. Therefore, you can compile and run the smaller files individually. Each subdirectory contains a **build** script that performs this logic automatically.

We define the compiler's name as **$PHCC** (or **%PHCC%**) in the **envsuite** scripts. Configure this to the name of your compiler's executable file (e.g. **mycc**).

You can record the reasons for each compile-time skipped case or run-time failure. In your **flags.h** file you can add a definition to some compile-time flags, such as

```
#define SKIP_131Y1_11 our parser error
#define FAIL_151Y11  Plum Hall bug?
```

Once you've categorized your skips and fails in this way, the strings you defined will show up in the execution output, something like this:

```
#SKIPPED _131Y1_11 (>our parser error<)
#FAILED _151Y11  (>Plum Hall bug?<)
```

And the "unexpected" skips and fails will show up with the distinctive string **"(><)"** attached to each "unexpected" skip or fail. This makes it much easier to re-run the test suite after you've made compiler changes, because you can quickly search for the **"><"** string in the output to see if any new failures have appeared.

The `buildmax` script will compile all *unbuilt* test cases using Make dependencies, where `buildall` will build all of the test cases regardless.

2.: CONFIGURATION

2.1 What You Need to Know and Do

In order to install and run ***Suite***++, there are several things you need to know, and several things you need to be able to do. If you don't have this knowledge yourself, then you need to locate someone who knows these things and is able to provide you with the information.

- You need to know how to use a text editor on each system you will be using.
- You need to know the basics of how to write and execute "script" (or "batch") files on each system.
- You need to know how much free disk space is available on each system. Fifty megabytes (50 MB) is often enough, if you remove each executable file after gathering its output. If you have less, refer to the Resources section later in this chapter for details.
- You need to know some C++ programming, to customize certain files and to understand the general meaning of the compiler diagnostics that may be produced by some of the nastier test cases.
- You need to know which compiler and library you are supposed to test, and what commands, arguments, environment settings, etc., are needed in order to invoke the compiler you're testing. (The compiler you're testing is called the target compiler.) You may also need to use a different compiler to compile the tool programs themselves. This is known as the host compiler, and it may have its own commands, arguments, environment settings, etc.
- Similarly, you need to know how to invoke the target linker and the host linker, to link the object-files produced by the compilers together with the appropriate libraries.

- Once the target compiler and target linker have produced an executable program to be tested, you need to know how to execute that executable program. On some systems this is almost trivial; on others it involves downloading from one machine to another, capturing output, networking the output back to the host machine, etc.

2.2 Running Suite++

There are many different modes in which you can use the Plum Hall Suites:

- Script (or "batch"') command files for compiler, linker, etc, or "line-by-line" individual commands.
- Host compiling (host and target compiler are the same), or cross compiling (host and target are different).
- UNIX platform, or Windows platform, or some other platform.

We have packaged the ***Suite***++ so that any set of these choices can be chosen.

2.3 Scripts

Using scripts (or "batch"' files) for compiler, linker, etc., simplifies many aspects of running the suite in varying environments. For example, many QA departments will need to routinely re-execute ***Suite***++ using dozens of different compiler flags and options. Using an unchanging set of compiler scripts, and just changing the flags and options in one script, or just setting the flags into environment variables, allows routine re-running of ***Suite***++.

In ***Suite***++. there is only one script to perform compile-link-and-go:

`xvsclgo` *pgm* **[***output-file-name***] [`bfile`]**

Compile *pgm*, taking source and headers from the appropriate directories. Put diagnostic messages into *pgm.clg*. Put output into *output-file-name,* if specified, otherwise send output to standard output. If the third argument is *bfile*, *pgm.cpp* will be linked with *pgm_b.cpp*.

envsuite

The `envsuite` script requires hand-configuration of environment variables for host and target compilers. You must examine it line-by-line. Here are a few of the environment variables it defines:

`PHCC`	the name of the target compiler
`PHCFLAGS`	compiler flags (for target compiler)
`OLDPATH`	original value of PATH variable before starting
`PATH`	command search path, including compilers, linkers, etc.
`PHCVS`	the directory where your CV-Suite is installed

UNIX CONSIDERATIONS

If you are on a UNIX platform, you may need to execute the **`chmodal`**l script:

`sh chmodall`

in order to mark all your script files as executable files. (It can't hurt, whether needed or not.)

The scripts and makefiles need three commands which are common on UNIX but not standard on Windows: **cat, rm** and **cp**. We have written work-alike C source files named **phcat.c** (for ``Plum Hall cat"), **phcp.c** (for ``Plum Hall cp"), and **phrm.c** (for "Plum Hall rm"). The **makefile** in **dst-win** will compile these to produce exe files (**phcat.exe, phcp.exe, phrm.exe**). After building each of these exe files, the **makefile** invokes a "setup" script (**setup-cat.bat, setup-cp.bat, setup-rm.bat**). Using "cat" as an example, the setup script determines whether a command named **cat** is already available on this system. If not, it copies **phcat.exe** to be named **cat.exe**, so that any further invocation of **cat** will invoke this exe file.

2.4 buildmax

When you have configured for your choices of environment, you should be ready to run the tests.

The **buildmax** command runs **xvsclgo** upon each of the source files in the **conform** directory. It also runs the appropriate tests from the CV-Suite. If you have not already read the CV-Suite manual, you should read it now.

The **buildmax** command also builds the summary files (`.sum,.det,.html` files), using the appropriate file of expected results (`.exp` file).

Besides the scripts, you will need to configure these other files that are in the destination directory:

flags.h	configurable parameters, including **SKIP** and **FAIL** flags
hocompil.h	characteristics of host-compiler (if different from target compiler)
homachin.h	characteristics of host-machine (if different from target machine)
hodefs.h	flags for hosted compilation (if different from **defs.h**)

Setting the envsuite environment

Each time you begin a testing session it is important to "source"' the **envsuite** script to establish all the necessary environment variables. This operation exports the environment variables into your interactive shell.

You do this in different ways depending on your host system's command processor or "shell":

MS-DOS	simply type **envsuite**.
Bourne shell	use the "dot"' command: **". ./envsuite"**

2.5 Installing This Release

We try to accommodate our customers' wide variety of environments, operating systems, and purposes for the suite. Also, we try to use update procedures which will be reasonably efficient for those who make no changes to the distributed Suite, while still being flexible enough for those of you who make local changes.

Some of you are primarily interested in the quality assurance process of running the suites, exactly as distributed, in a reliable fashion that takes a minimum of your time. Others of you are developing compilers that change daily, tracking the latest Standard, with numerous local changes and **SKIP** flags to accommodate unimplemented features.

We always welcome ideas and suggestions for improvement, so please let us know if you see a better way of doing something.

Minimal and Complete Installation Choice

The original packaging of ***Suite***++ contained only a few dozen source files. Although the small number of compilations was convenient, the downside was the iterative manual process of determining the **SKIP** flags for the **flags.h** header, to skip language features yet unimplemented in the compiler.

Installing the Distribution

Most importantly, install to an empty directory. Installing over the old directory structure will cause no end of chaos. (Also, removing the prior release will help you fulfill your license requirement to maintain source-control of previous versions.)

Verifying Your Files

No matter which method you used for updating—diskette, tape, or patch from diffs—you can check your resulting updated files by compiling the **txtchk** program, and then using it to test the checksum of all your file contents:

```
cd ~/PlumHall/xvs22a-gcc-c20  (or whatever your source root is named)
txtchk -f xvs22a
```

3.: CONFORM

The CONFORM section provides several thousand C++ programs, each covering part of a clause in the Language section of the Standard:

t02a *Clause 2;*
t03a *First part of clause 3;*
t03b ***s****econd part of clause 3;*
etc.

Each program writes a report to its **.out** file in a form very similar to the output of the **LANG** program in the C Suite. That is, **t02a** reports that it has executed the first test with the output

```
***** Reached first test *****
```

t02a reports errors using messages of the form

```
ERROR in t02a at line 656: (4) != (5)
```

and prints a summary of the form

```
***** 18 individual successful items in t02a
***** 11 successful tests in t02a
***** 0 errors detected in t02a
***** 0 skipped sections in t02a
```

An “individual successful item” is the successful outcome of one individual test function (**`ieq`**, **`chk`**, etc.). A “successful test” is the completion of a **`begin_case-end_case`** sequence with no errors in its individual items.

3.1 Compiling and Executing Suite++ CONFORM

In the distribution, C++ source files have a **.cpp** extension and headers have a **.h** extension. The **.c** files may be renamed to, say, **.cxx** files to suit your compiler, but the **.h** files should not be renamed.

The **buildmax** script specifies all the steps for building and executing each program. Or you can create the executables by invoking your compiler and linker directly from a command line.

For example, to create the executable for **t02a**, compile and link the following files: **t02a.cpp** and **util.c**. The compiler command line is typically of the form

```
CC -ot02a t02a.cpp util.c
```

where **CC** is the C++ compiler command, **t02a.cpp** supplies the main function for the program, and **util.c** contains utility functions used in the test cases.

Alternatively, if in your **flags.h** file you place a definition like

```
#define UTIL_SHOULD_BE_INCLUDED
```

then the compilation will **#include "util.c"** and you need not link with it. This simplifies the compilation process somewhat, and if the compiler supports precompiled headers there is not much overhead in the method.

3.2 Selective Enabling/Disabling with flags.h

In the destination directory, you should create a file named **flags.h**. This header is **#included** by each ***Suite***++ file, so that you can record specific enable/disable flags for the tests being made in this directory tree.

If your C++ compiler cannot compile a particular test case, you can use a **SKIP** flag to disable that case. For example, to prevent compilation of test case **_17312Y21** in **t02a.c**, add

```
#define SKIP_17312Y21   because some reason
```

to the file **flags.h**. Then recompile and relink **t02a**. The line

```
#SKIPPED: _17312Y21 (>because some reason<)
```

will appear in the output when you execute **t02a**. The total number of skipped cases appears at the end of the output.

You can also define **DISALLOW** flags in **flags.h** to globally disable certain language features that your compiler may not be able to handle.

For example

```
#define DISALLOW_MEMBER_TEMPLATES
```

compiles alternative code for some cases to accommodate the absence of member templates. See the **flags.h** file for description of each flag. (But note that the **DISALLOW** flags do not affect **negtests.**)

Using

```
#define DISALLOW_CXX17
```

causes the set of all those test cases to be disabled.

For testing strict conformance, these "DISALLOW" flags are required, and they are all turned on (in "flags.h")

The "disputed cases", described in the following section, are excluded by default.

3.3 Controversial Cases

We strive to make ***Suite***++ test the C++ language as commonly understood by the worldwide C++ community. The purpose of the ongoing standard is to capture that understanding. However, there will probably always be specific issues in the language which evoke differing interpretations, and hence there will probably be specific tests Suite++ which evoke differing opinions from ***Suite***++ users about the expected results. ***Suite***++ accommodates controversial tests in the category of "disputed cases", which are disabled by

```
#define DISALLOW_DISPUTED
```

A "disputed case" is a test for which some ***Suite***++ users have expressed the view that, although the test reflects the words in the Standard, the words in the Standard do not reflect common practice, or that the words in the Standard are in the process of being revised within the C++ committee.

We hope to eventually resolve all the disputed cases and convert them to agreeable tests in the **CONFORM** sections of ***Suite***++. We welcome your feedback regarding our judgments in these areas.

3.4 Running the CONFORM Programs

Once configuration is completed, you are ready to compile and execute the **CONFORM** programs. Any compile errors reported may represent currently-unimplemented syntactic features, or bugs in your compiler, or bugs in ***Suite***++. Or, don't forget, sometimes a compile error means that the compiler wasn't properly installed, or that you weren't told the proper command-line options to use, or that the compilation environment wasn't properly set up. You have to investigate all these possibilities.

If you are unable to trace the cause of any compile errors whilst building **CONFORM**, you should contact Plum Hall for assistance.

Users have asked us to help speed up testing on parallel multiprocessor systems. We have provided a **makefile** in the **dst*/conform** folder, which can be executed with

```
make -k --jobs N all
```

so that users with N processors should see an N-fold speedup.